Self-Organization in Multi Agent Systems: a Middleware Approach

Marco Mamei
Università di Modena e Reggio Emilia
Via Allegri 13, Reggio Emilia ITALY
mamei.marco@unimo.it

Franco Zambonelli
Università di Modena e Reggio Emilia
Via Allegri 13, Reggio Emilia ITALY
franco.zambonelli@unimo.it

ABSTRACT

Self-organization is built upon two main building blocks: adaptive and uncoupled interaction mechanisms and context-awareness. Here we show how the middleware TOTA (Tuples On The Air) supports self-organization by providing effective abstractions for the above two building-blocks. TOTA relies on spatially distributed tuples for both supporting adaptive and uncoupled interactions between agents, and context-awareness. Agents can inject these tuples in the network, to make available some kind of contextual information and to interact with other agents. Tuples are propagated by the middleware, on the basis of application specific patterns, defining sorts of "computational fields", and their intended shape is maintained despite network dynamics, such as topological reconfigurations. Agents can locally "sense" these fields and can rely on them for both acquiring contextual information and carrying on distributed self-organizing coordination activities. Several application examples in different scenarios show the effectiveness of our approach.

Keywords

Self-Organization, Agents, Coordination, Context Awareness, Middleware.

1. INTRODUCTION

IT scenarios, at all levels, are witnessing a radical change: from systems constituted by components fixedly and strictly coupled at design time, to systems based on autonomous, uncoupled and transiently interacting components [ZamP02].

In computer science, application have always been built by adopting programming paradigms rooted on strictly coupled, not-autonomous components. Following this approach, components are coupled at design time by fixed interaction patterns. Although simple, this approach turned out to be really brittle and fragile, not being able to cope with reconfiguration and faults. Only in recent years, the research on software agents has fostered new programming paradigms based on autonomous components (i.e. components with a separated thread of execution and control) interacting to realize an application [BelPR01, CabLZ02, PicMR01].

This shift of paradigm is well motivated by the robustness, scalability and flexibility of these systems: if a component breaks down, the others can re-organize their interaction patterns to account for such a failure, if new components are added to the system, they can discover who is around and start to interact with them. The key element leading to such robust, scalable and flexible behaviors is self-organization. Autonomous components must be able to self-organize their activities' patterns to achieve goals, that possibly exceed their capabilities as singles, despite - and possibly taking advantage - of environment dynamics and unexpected situations [ParBS02, SerR02, ZamM02]. Nature, for example, "adopts" these ideas at all scales (e.g. in social insects like ants, and in cells like in the immune system) [Bar97, Bar02, BonDT99].

The first signs of these concepts can be found in modern distributed computing where the inherent dynamism of networks (e.g. delays and links unavailability) forces distributed applications' components to autonomously adapt and self-organize their behavior to such dynamism. On the one hand, Internet applications, traditionally built following a client-server approach, are gradually replaced by their peer-to-peer (P2P) counterpart. By investing on peers autonomy, P2P applications can self-organize their activities to achieve unprecedented levels of robustness and flexibility (e.g. peers can dynamically discover communication partners and autonomously engage, also third-party, interaction patterns). On the other hand, components' autonomy and self-organization is at the basis of ubiquitous and pervasive computing, where intrinsic mobile components are connected in wireless, amorphous networks. In such a dynamic scenario, in fact, agents have to constantly rearrange and self-organize their activities to take in account the ever changing environment.

Unfortunately, we still do not know how to program and manage these kind of autonomous self-organizing systems. The main conceptual difficulty is that we have direct engineered control only on agents' local activities, while the application task is often expressed at the global scale [Bar02, BonDT99]. Bridging the gap between local and global activities is not nearly easy, but it is although possible: distributed algorithms for autonomous sensor networks have been proposed and successfully verified, routing protocols is MANET (in which devices coordinate to let packets flow from sources to destinations) have been already widely used. The problem is still that the above successful approaches are ad-hoc to a specific application domain and it is very difficult to generalize them to other scenarios. There is a great need for general, widely applicable engineering methodologies, middleware and APIs to embed, support and control self-organization in multiagent systems [Abe00, SerD02, KepC03, ZamP02]. From our point of view, self-organization is a sort of distributed coordination and its main building blocks are those constituting the core of agents' autonomy: *(i)* adaptive and uncoupled interaction mechanisms and *(ii)* context-awareness. With regard to the first point, environment dynamism and transiently connected components call for flexible, adaptive and uncoupled interactions. Moreover, by its own nature, coordination requires context-awareness. In fact, an agent can coordinate with other agents only if it is somehow aware of "what is around", i.e., its context. However, when agents are embedded in a possibly unknown, open and dynamic environment (as it is in the case of most pervasive computing scenarios), they can hardly be provided with enough a priori up-to-date contextual knowledge. Starting from these considerations, it is fundamental to provide agents with simple, easy to be obtained, and effective contextual information, supporting and facilitating their coordination activities in a robust and adaptive way.

The contribution of this paper is to show how the abstractions promoted by a novel middleware infrastructure called TOTA ("Tuples On The Air"), suit the need of self-organization. Coherently with the above considerations, the key objective of TOTA is to define a single abstraction both to: *(i)* promote uncoupled and adaptive interactions; and *(ii)* provide agents with simple, yet expressive, contextual information to actively support adaptivity, by discharging application components from the duty of dealing with network and application dynamics. To this end, TOTA relies on spatially distributed tuples, to be injected in the network and propagated accordingly to application-specific patterns. On the one hand, tuple propagation patterns are dynamically re-shaped by the TOTA middleware to implicitly reflect network and applications dynamics, as well as to reflect the evolution of coordination activities. On the other hand, application agents have simply to locally "sense" tuples to acquire contextual information, to exchange information with each other, and to implicitly and adaptively orchestrate their coordination activities. To take a metaphor, we can imagine that TOTA propagates tuples in the same way as the laws of nature provides propagating fields in the physical space: although particles do not directly with each other and can only locally perceive such fields, they exhibit globally orchestrated and adaptive motion patterns.

The following of this paper is organized as follows. Section 2 overviews the TOTA approach. Section 3 details the architecture of the TOTA middleware, its main underlying algorithm and its implementation. Section 4 describes some application examples, showing how can TOTA be effectively applied to different scenarios. Section 5 discusses related works. Section 6 concludes and outlines future works.

2. TUPLES ON THE AIR: OVERVIEW

The driving objective of our approach is to address together the two requirements introduced at the beginning of the previous section (uncoupled and adaptive interactions and context-awareness), by exploiting a unified and flexible mechanism to deal with both context representation and agents' interactions, and thus also leading to a simpler, and lighter to be supported, applications.

In TOTA, we propose relying on distributed tuples for both representing contextual information and enabling uncoupled interaction among distributed application agents. Unlike traditional shared data space models, tuples are not associated to a specific node (or to a specific data space) of the network. Instead, tuples are injected in the network and can autonomously propagate and diffuse in the network accordingly to a specified pattern (see Figure 1). Thus, TOTA tuples form a sort of spatially distributed data structure able to express not only messages to be transmitted between application components but, more generally, some contextual information on the distributed environment.

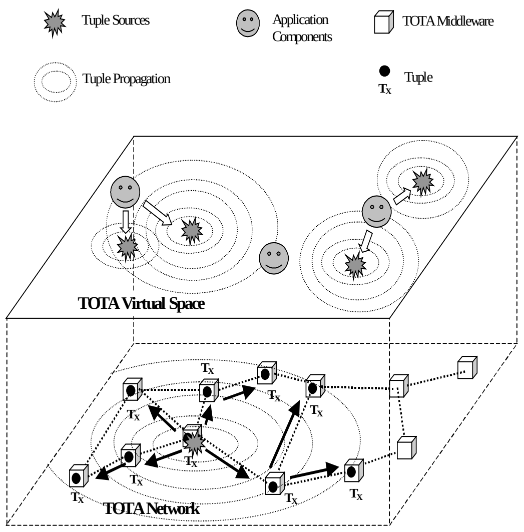

Figure 1: The General Scenario of TOTA: application components live in an environment in which they can inject tuples that autonomously propagate and sense tuples present in their local neighborhood. The environment is realized by means of a peer-to-peer network in which tuples propagates by means of a multi-hop mechanism.

To support this idea, TOTA is composed by a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. Each TOTA node holds references to a limited set of neighboring nodes. The structure of the network, as determined by the neighborhood relations, is automatically maintained and updated by the nodes to support dynamic changes, whether due to nodes' mobility or to nodes' failures. The specific nature of the network scenario determines how each node can found its neighbors: e.g., in a MANET scenario, TOTA nodes are found within the range of their wireless connection; in the Internet they can be found via an expanding ring search (the same used in most Internet peer-to-peer systems [Rat01]).

Upon the distributed space identified by the dynamic network of TOTA nodes, each component is capable of locally storing tuples and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop accordingly to their propagation rule, eventually leaving copies of itself at every propagation step. In fact, a TOTA tuple is defined in terms of a "content", and a "propagation rule".

$$T=(C,P)$$

The content C is an ordered set of typed fields representing the information carried on by the tuple. The propagation rule P determines how the tuple should be distributed and propagated in the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rules can determine how tuple's content should change while it is propagated. Tuples are not necessarily distributed replicas: by assuming different values in different nodes, tuples can be effectively used to build a distributed overlay data structures expressing some kind of contextual and spatial information. On a different perspective, we can say that TOTA enrich a network with a notion of space. A tuple incrementing one of its fields as it gets propagated identifies a sort of "structure of space" defining the network distances from the source. By relying on data acquired by proper physical localization devices, like GPS systems or beacon-based triangulation, tuples can provide a structure of space based on the actual physical location of devices and thus enabling a tuple to be propagated, say, at most for 10 meters from its source. Taking this approach to the extreme, one could think at mapping the peers of a TOTA network in any sort of virtual overlay space [Rat01], and propagating tuples in such virtual space.

The spatial structures induced by tuples propagation must be maintained coherent despite network dynamism. To this end, the TOTA middleware supports tuples propagation actively and adaptively: by constantly monitoring the network local topology and the income of new tuples, the middleware automatically re-propagates tuples as soon as appropriate conditions occur. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to nodes' movements, the distributed tuple structure automatically changes to reflect the new topology. For instance, Figures 2, 3, and 4, show how the structure of a distributed tuple can be kept coherent by TOTA in a MANET scenario, despite dynamic network reconfigurations.

From the application agents' point of view, executing and interacting basically reduces to inject tuples, perceive local tuples, and act accordingly to some application-specific policy. Software agents execute on a node, in which the TOTA middleware has been installed. They can inject new tuples in the network, defining their content and their propagation rule. They have full access to the local content of the middleware (i.e., of the local tuple space), and can query the local tuple space – via a pattern-matching mechanism – to check for the local presence of specific tuples. In addition, TOTA provides agents with a virtual global, read-only, view of the tuple spaces of one-hop TOTA neighbors. This is basically a virtual tuple space consisting in a union of all the TOTA tuple spaces in the one-hop neighborhood [PicMR01]. Looking at this view an agent can see the tuples stored in its closest neighborhood. Finally, agents can be notified of locally occurring events (i.e., changes in tuple space content and in the structure of the network neighborhood). In TOTA there is not any primitive notion of distributed query. Still, it is possible for an agent to inject a tuple in the network and have such distributed tuple be interpreted as a query at the application-level, by having other agents in the network react to the income of such tuple, i.e., by injecting a reply tuple propagating towards the enquiring node.

The overall resulting scenario – making it sharp the analogy with the physical world anticipated in the introduction – is that of applications whose agents: *(i)* can influence the TOTA space by propagating application-specific tuples; *(ii)* execute by being influenced in both their internal and coordination activities by the locally sensed tuples; and *(iii)* implicitly tune their activities to reflect network dynamics, as enabled by the automatic re-shaping of tuples' distributions of the TOTA middleware.

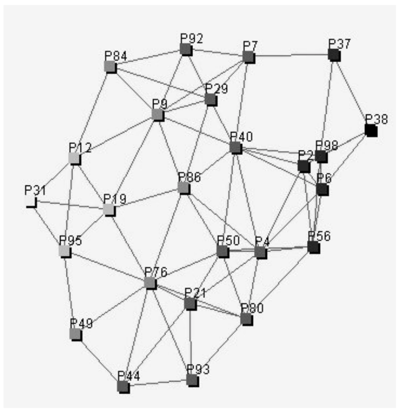

Figure 2: P31 propagates a tuple that increases its value by one at every hop. Tuple hop-value is represented by node darkness.

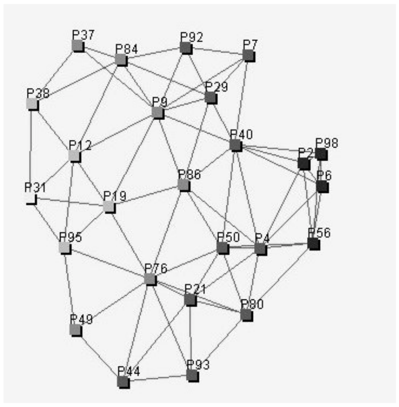

Figure 3: P37 and P38 moves closer to the source and their tuples automatically change values to maintain the distributed tuple coherency.

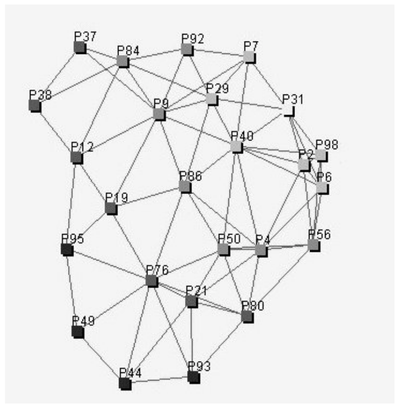

Figure 4: When the tuple source P31 moves, all tuples are updated to take into account the new topology.

3. TOTA MIDDLEWARE

3.1 Tota Architecture

As introduced in the previous section, a network of possibly mobile nodes running each one a TOTA middleware constitutes the scenario we consider. Each TOTA node holds references to neighboring nodes and it can communicate directly only with them. While in an ad-hoc network scenario it is rather easy to identify the node's neighborhood with the range of the wireless link (e.g. all the nodes within 10m, for a Bluetooth wireless link), in a wired scenario like the Internet is less trivial. We imagine however that in such a case the term is not related to the real reachability of a node, but rather on its addressability (a node can communicate directly with another only if it knows other node's IP address). This means that at the very bottom of the TOTA middleware there is a system to continuously detect neighboring nodes and to store them in an appropriate list. In a MANET scenario this system is directly connected to the wireless network and detects in-range nodes. In a wired scenario, like the Internet, it can start an expanding-ring search for other TOTA nodes, or it can simply query a central repository (e.g. a known web-site) and download a list of TOTA nodes' IP addresses.

Each TOTA middleware is provided with a local tuple space to store the tuples that reached that node during their propagation. Agents can access the local tuple space via Linda-like operations [GelC92]. Moreover, the TOTA middleware offers a virtual global, read-only, view of the tuple spaces of one-hop TOTA neighbors. Looking at this view an agent can see the tuples stored in its closest neighborhood. This feature is fundamental since the main TOTA algorithms require the knowledge of tuples present in at least a one-hop neighborhood (see 3.3 and 5).

As stated in the previous section, each TOTA middleware is in charge to store, propagate and keep updated the tuples' structure. To achieve this task TOTA needs a mean to uniquely identify tuples in the system in order for example to know whether a particular tuple has been already propagated in a node or not. Tuple's content cannot be used for this purpose, because the content is likely to change during the propagation process. To this end, each tuple will be marked with an *id* (invisible at the application level) that will be used by TOTA during tuples' propagation and update to trace the tuple. Tuples' *id* is generated by combining a unique number relative to each node (e.g., the MAC address) together with a progressive counter for all the tuples injected by the node.

With regard to the tuples' propagation rule, in order to give full flexibility to the model, the propagation rule can be an arbitrary piece of code. Upon the receipt of a tuple, the middleware invoke its propagation method in a call-back fashion. In its propagation algorithm the tuple has full access over the TOTA API, thus it can take decisions on the basis of the already stored tuples or on the basis of the network local topology. Most importantly a tuple can subscribe to events, in the same way as agents installed upon the TOTA middleware. This is extremely important, because it allow a tuple to remain live and able to react to changes in its environment. This fact will be at the core of the tuples' maintenance operations described in 3.3.

From the architecture point of view, the TOTA middleware is constituted by four main parts (see Figure 5): *(i)* the TOTA API, is the main interface between the application and the middleware. It provides functionalities to let the application to inject new tuples in the system, to access the local tuple space, or to place subscriptions in the event interface. *(ii)* The EVENT INTERFACE is the component in charge of asynchronously notifying the application about subscribed events, like the income of a new tuple or about the fact a new node has been connected/disconnected to the node's neighborhood. *(iii)* The TOTA ENGINE is the core of TOTA: it is in charge of maintaining the TOTA network by storing the references to neighboring nodes and to manage tuples' propagation by opening communication sockets to send and receive tuples. This component is in charge of sending tuples injected from the application level, and to apply the propagation rule of received tuples and to re-propagate them accordingly. Finally this component monitors network reconfiguration and the income of new tuples and updates and re-propagates already stored tuples to maintain tuples' structure coherency. *(iv)* The VIRTUAL TUPLE SPACE is the component offering a virtual global view of the tuple spaces of one-hop TOTA neighbors.

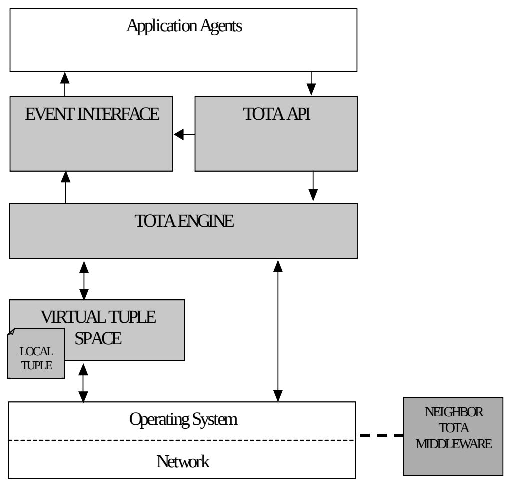

Figure 5: TOTA Middleware

3.2 Implementation

From an implementation point of view, we developed a first prototype of TOTA running on laptops and on Compaq IPAQs equipped with 802.11b and Personal Java. IPAQ connects locally in the MANET mode (i.e. without requiring access points) creating the skeleton of the TOTA network. Tuples are being propagated through multicast sockets to all the nodes in the one-hop neighbor. The use of multicast sockets has been chosen to improve the communication speed by avoiding 802.11b unicast handshake. By considering the way in which tuples are propagated, TOTA is very well suited for this kind of broadcast communication. We think that this is a very important feature, because it will allow, in the future, implementing TOTA also on really simple devices (e.g. micro sensors) that cannot be provided with sophisticate communication mechanisms [Loo01]. Other than this communication mechanism, at the core of the TOTA middleware there is a simple event-based engine. This component is able to collect subscriptions of interesting events and to invoke reactions on the subscribed agents, in a call-back fashion.

Actually we own only a dozen of IPAQs and laptops on which to run the system. Since the effective testing of TOTA would require a larger number of devices, we have implemented an emulator to analyze TOTA behavior in presence of hundreds of nodes. The emulator, developed in Java, enables examining TOTA behavior in a MANET scenario, in which nodes topology can be rearranged dynamically either by a drag and drop user interface or by autonomous nodes' movements. The strength of our emulator is that, by adopting well-defined interfaces between the emulator and the application layers, the *same* code "installed" on the emulated devices can be installed on Personal Java real devices (e.g. Compaq IPAQs) enabled with wireless connectivity. This allow to test application first in the emulator, then to transfer them directly in a network of real devices. In order to rend the emulated scenario as close as possible to the real scenario, devices' battery consumption and wireless network glitches have been emulated as well. The snap-shots of Figure 2, 3, and 4 are actually rendered via the implemented emulator.

3.3 Tuples Propagation and Maintenance

As stated above, the main functionality offered by TOTA is the mechanism to propagate distributed tuples and to maintain their intended shape despite changes in network topology.

While tuple propagation is simple, since it basically consists in an epidemic communication schema, in which tuples are propagated hop-by-hop, following a breadth-first pattern, tuples maintenance is much more complex. Maintenance operations are mainly required upon a change in the network topology, to have the distributed tuples reflect the new network structure. This means that maintenance operations are possibly triggered whenever, due to nodes' mobility or failures, new links in the network are created of removed. Because of scalability issues, it is fundamental that the tuples' maintenance operations are confined to an area neighboring the place in which the network topology had actually changed. This means that, if for example, a device in a MANET breaks down (causing a change in the network topology) only neighboring devices should change their tuples' values. The size of this neighborhood is not fixed and cannot be predicted a-priori, since it depends on the network topology.

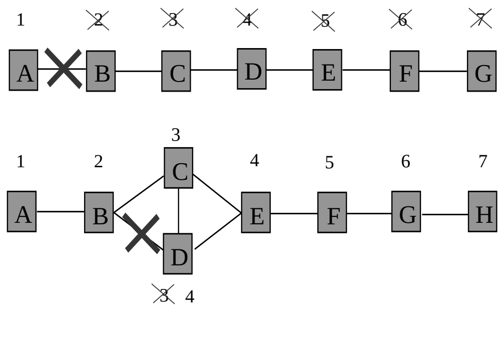

Figure 6: The size of the update neighborhood depends on the network topology. Here is an example with a tuple incrementing its integer content by one, at every hop, as it is propagated far away from its source (top) the specific topology force update operations on the whole network (bottom) if alternative paths can be found, updates can be much more localized.

For example, if the source of a tuple gets disconnected from the rest of the network, the updates must inevitably involve all the other peers in the network (that must erase that tuple form their repositories, see figure 6-top). However, especially for dense networks, this is unlikely to happen, and usually there will be alternative paths keeping up the tuple shape (see figure 6-bottom).

How can we perform such localized maintenance operations in a fully distributed way? To fix ideas, let us consider the case of a tuple incrementing its integer content by one, at every hop, as it is propagated far away from its source.

Given a local instance of such a tuple *X*, we will call *Y* a *X*'s *supporting tuple* if: *Y* belongs to the same distributed tuple as *X*, *Y* is one-hop distant from *X*, *Y* value is equal to *X* value minus 1.With such a definition, a *X*'s supporting tuple is a tuple that could have created *X* during its propagation.

Moreover, we will say that *X* is in a *safe-state* if it has a supporting tuple, or if it is the source of the distributed tuple. We will say that a tuple is not in a safe-state if the above condition does not apply.

Each local tuple can subscribe to the income or the removal of other tuples belonging to its same type in its one-hop virtual tuple space. This means, for example, that the tuple depicted in figure 6-bottom, installed on node F and having value 5 will be subscribed to the removal of tuples in its neighborhood (i.e. nodes E and G).

Upon a removal, each tuple reacts by checking if it is still in a safe-state. In the case a tuple is in a safe-state, the tuple the removal has not any effect - see later -. In the case a tuple is not in a safe state, it erases itself from the local tuple space. This eventually cause a cascading tuples' deletion until a safe-state tuple can be found, or the source is eventually reached, or all the tuples in that connected sub-network are deleted (as in the case of figure 6-top).

When a safe-state tuple observe a deletion in its neighborhood it can fill that gap, and reacts by propagating to that node. This is what happens in figure 6-bottom, safe-state tuple installed on mode C and having value 3 propagates a tuple with value 4 to the hole left by tuple deletion (node D). It is worth noting that this mechanism is the same enforced when a new peer is connected to the network.

Similar considerations applies with regard to tuples' insertion: when a tuple sense the arrival of a tuple having value lower than its supporting tuple, it means that, because of nodes' mobility, a short-cut leading quicker to the source happened. Also in this case the tuple must update its value to take into account the new network topology.

How many information must be sent to maintain a the shape of a distributed tuple? What is the impact of a local change in the network topology in real scenarios? To answer these questions we exploited the implemented TOTA emulator, being able to derive results depicted in figure 7.

The graph shows the results of three experiments, conducted on different networks. We considered networks having an average density (i.e. average number of nodes directly connected to an other node) of 4, 6 and 8. In each network, a tuple, incrementing its content at every hop, had been propagated. Nodes in the network move randomly, continuously changing the network topology. The number of messages sent between peers to keep the tuple shape coherent had been counted. Messages exchanged by peers one-hop away form the peer that caused the topology change are added together, and so on for peers two-hop away, three hop-away, etc. These values are depicted in figure 7.

The most important consideration we can make looking at the graph, is that, upon a connection, a lot of update operations will be required near the source of the topology change, while only few operations will be required far away from it. This implies that, even if the TOTA network and the tuples being propagated have no artificial boundaries, the operations to keep their shape consistent are strictly confined within a locality scope. This fact supports the feasibility of the TOTA approach in terms of its scalability. In fact, this means that, even in a large network with a lot of nodes and tuples, we do not have to continuously flood the whole network with updates, eventually generated by changes in distant areas of the network. Updates are confined within a locality scope from where they took place.

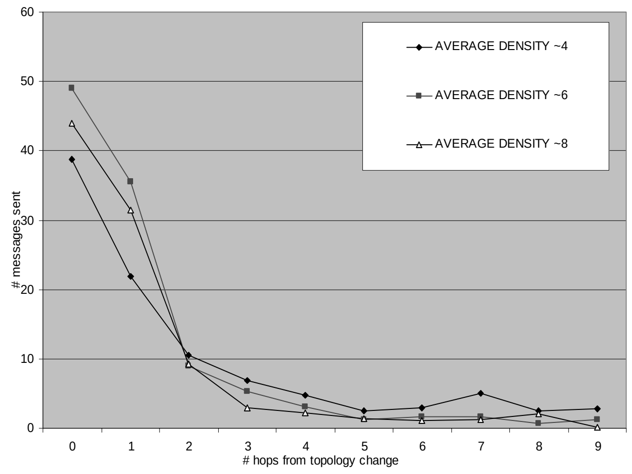

Figure 7: Experimental results: locality scopes in tuple's maintenance operations emerge in a network without predefined boundaries.

4. APPLICATION EXAMPLES

In this section, to prove the generality of our approach, we will show how to exploit TOTA to solve several problems typical of dynamic network scenarios, by simply implementing different tuples' propagation rules.

4.1 Motion Coordination

To show the capability of achieving globally coordinated behaviors with TOTA, we focus on a specific instance of the general problem of motion coordination. Motion coordination has been widely studied in several research areas: robotics, simulations, pervasive computing, multi agent systems, etc. Among the others, a particularly interesting and successful approach is the one that exploit the idea of potential fields to direct the movement of the involved entities [SheS02, MamLZ02]. As a first example, we will consider the problem of letting a group of mobile components (e.g., users with a PDA or robots) move maintaining a specified distance from each other. To this end, we can take inspiration from the mechanism used by birds to flock [BonDT99]: flocks of birds stay together, coordinate turns, and avoid each other, by following a very simple swarm algorithm. Their coordinated behavior can be explained by assuming that each bird tries to maintain a specified separation from the nearest birds and to match nearby birds' velocity. To implement such a coordinated behavior in TOTA, each component can generate a tuple *T=(C,P)* with following characteristics:

C= (FLOCK, nodeName,val)

P= ("val" is initialized at 2, propagate to all the nodes decreasing by one in the first two hops, then increasing "val" by one for all the further hops)

Thus creating a distributed data structure in which the *val* field assumes the minimal value at specific distance from the source (e.g., 2 hops). This distance expresses the intended spatial separation between components in the flock. To coordinate movements, components have simply to locally perceive the generated tuples, and to follow downhill the gradient of the *val* fields. The result is a globally coordinated movement in which components maintain an almost regular grid formation by clustering in each other *val* fields' minima.

To test the above coordination mechanism we used the emulator: the snap-shots of Figure 8 shows a MANET scenario in which a group of four components (in black) proceeds in a flock, maintaining a one hop distance. The other nodes in the network remain still and just store and forward flocking tuples.

Another interesting example of motion coordination, regard the problem of letting mobile users to meet somewhere. Here we can imagine that each member of the meeting group injects a tuple with the following characteristics:

C= (MEET, nodeName, val)

P= ("val" is initialized at 0, propagate to all the nodes increasing "val" by one for all the further hops)

By relying on this tuple, we can realize different meeting policies:

1. The group of users wants to meet in the point where member *x* is located. This is the simplest case and each user can move by following downhill the tuple having in its content *x* as *nodeName*. It is interesting to notice that this approach works even if person *x* moves after the meeting has been scheduled. The meeting will be automatically rescheduled in the new minimum of *x*'s tuple.
2. The group of users wants to meet in the point that is between them (their "barycenter"). To this purpose each user *i* can follow downhill a linear combination of all the other MEET tuples. In this way all the users "fall" towards each other, and they meet in the point that is in the middle. It is interesting to notice, that this "middle point" is evaluated dynamically and the process takes into consideration the crowd or unexpected situations. So if some users encounter a crowd in their path, the meeting point is automatically changed to one closer to these unlucky users.

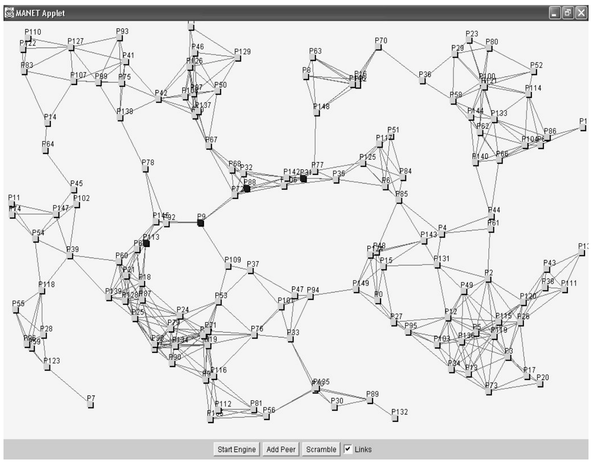

Figure 8. Flocking in the TOTA Emulator. Cubes are the nodes of a mobile ad-hoc network, with arcs connecting nodes in range with each other. Black cubes are involved in flocking, moving by preserving a 2 hop distance from each other.

4.2 Modular Robot Control

Another interesting application scenario, is in the control of a modular robot [YimZD02]: a collection of simple autonomous actuators with few degrees of freedom connected with each other. A distributed control algorithm is executed by all the actuators that coordinate to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). Currently proposed approaches [SheS02] adopts the biologically inspired idea of hormones to control such a robot. Hormone signals are similar to content based messages, but have also the following unique properties: they propagate through the network without specific destinations, their content can be modified during propagation and they may trigger different actions for different receivers. The analogies between hormones and TOTA tuples are evident and, in fact, we were able to easily implement a similar control algorithm on top of TOTA. The algorithm has been tested on the 3D modular robot simulator available at [Polybot]. Following the approach proposed in [SheS02], we will consider the implementation of a caterpillar gait on a chain-typed modular robot, composed by actuators having a single motorized degree of freedom (see figure 10). Each robot node (i.e. actuator) will be provided with a TOTA middleware, and with an agent driving its motor. In particular the head agent, the tail agent and the body agents will drive the head module, the tail module and the body modules respectively.

The head agent starts the movement by injecting a *caterpillar-tuple*. The tail agent injects the *gait-tuple*, upon the receipt of a new *caterpillar-tuple*.

The gait-tuple is simply a tuple notifying that the gait has been completed, it simply propagates from the tail to the head (i.e. it has a broadcast propagation rule) without storing.

The caterpillar-tuple has the following structure:

C = (state, angle)
P = (propagate hop-by-hop, storing on intermediate nodes changing the content accordingly to the table in figure 9. If on the head node and upon the receipt of a gait-tuple, re-apply propagation)

Each agent, upon the receipt of the caterpillar tuple, will drive the motor of its actuator to the angle in the content of the tuple. The coordination induced by the tuple leads the robot to the caterpillar gait as described in figure 10.

Current state	*New state*	*New angle*
At the beginning	*A*	*+45 deg*
A	*B*	*+45 deg*
B	*C*	*-45 deg*
C	*D*	*-45 deg*
D	*A*	*+45 deg*

Figure 9. Caterpillar tuple, propagation rule.

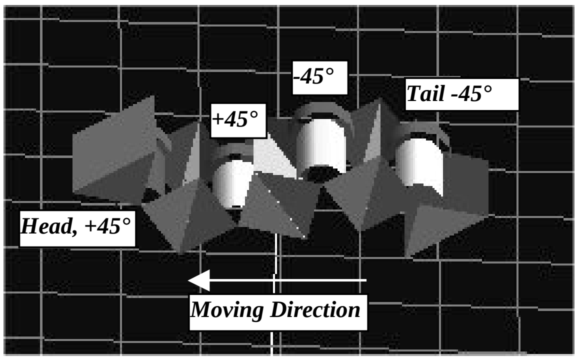

Figure 10. Caterpillar gait, in a chain-typed modular robot, composed of four actuators.

4.3 Ant-Based Routing on Mobile ad Hoc Networks

Routing protocols in wireless ad-hoc networks, inspired to the way in which ants collect food, have recently attracted the attention of the research community [BonDT99, Poo00]. Following this inspiration, the routing protocols build a sort of routing overlay structure (similar to ants' pheromone trials) by flooding the network and then exploit this overlaid structure for a much finer routing. We will show in this section how the basic mechanism of creating a routing overlay structure and the associated routing mechanism (similar to the ones already proposed in the area) can be effectively done within the TOTA model. The basic idea of the routing algorithm we will try to implement is the following [Poo00]: when a node X wants to send a message to a node Y it injects a tuple representing the message to be sent, and a tuple used to create an overlay routing structure, for further use.

The tuple used to create the overlay structure can be described as follows:

C=("structure", nodeName, hopCount)
P=(propagate to all the nodes, increasing hopCount by one at every hop)

The tuple used to convey the message will be:

C=("message", sender,receiver,message)

P=(if a structure tuple having my same receiver can be found follow downhill its hopCount, otherwise propagate to all the nodes)

This routing algorithm is very simple: *structure* tuples create an overlay structure so that a *message* tuple following downhill a *structure* tuple's *hopCount* can reach the node that created that particular structure. In all situations in which such information is absent, the routing simply reduces to flooding the network. Although its simplicity, this model captures the basic underling model of several different MANET routing protocols [Bro98]. The basic mechanism described in this section (tuples defining a structure to be exploited by other tuples' propagation) is fundamental in the TOTA approach and provides a great flexibility. For example it allows TOTA to realize systems such as CAN [Rat01] and Pastry [RowD01], to provide content-based routing in the Internet peer-to-peer scenario. In these models, peers forming an unstructured and dynamic community need to exchange data and messages not on the basis of the IP addressing scheme, but rather on the basis of the content of messages (e.g., "I need the mp3 of Hey Jude, no matter who can provide it to me"). To this end, these systems propose a communication mechanism based on a publish-subscribe model and rely on a properly built overlay space. A peer publishes information by sending them to a particular point of the overlaid space, while another read such information by looking for it in the same point of space (typically the process involves a hash function shared between all the peers, that maps keywords, associated to the information content, to points in space). TOTA can realize such systems by using a first layer of tuples defining the overlay space and then other tuples whose propagation rules let the tuples propagate efficiently in the overlaid space.

5. RELATED WORK

Several proposals in the last years are challenging the traditional ideas and methodologies of software engineering and inspired to physical, biological models are entering in the distributed application and multi agent system research frameworks.

An area in which the problem of achieving context-awareness and adaptive coordination has been effectively addressed (and that, consequently, has partially influenced our proposal) is amorphous and paintable computing [But02, Nag03]. The particles constituting an amorphous computer have the basic capabilities of propagating sorts of computational fields in the network, and to sense and react to such fields. In particular, particles can transfer an activity state towards directions described by fields' gradients, so as to make coordinated patterns of activities (to be used for, e.g. self-assembly) emerge in the system independently of the specific structure of the network (which is, by definition, amorphous). Similarly with TOTA, such an approach enables, via the single abstraction of fields, to both diffuse contextual information and to organize adaptive global coordination patterns. The main difference between TOTA and this approach is the application domain: TOTA is not only addressed to amorphous networks of nano- or micro-devices, but it aims also to address networks of mobile devices like cellular phones, PDA and laptops. Moreover, because of this difference, one of the TOTA main concerns, that is totally neglected in amorphous computer, is the need to constantly manage distributed tuples' values so as to maintain their intended shape despite network reconfigurations.

Anthill [BabM02] is a framework built to support design and development of adaptive peer-to-peer applications, that exploits an analogy with biological adaptive systems [BonDT99, ParBS02]. Anthill consists of a dynamic network of peer nodes, each one provided with a local tuple space ("nest"), in which distributed mobile components ("ants") can travel and can indirectly interact and cooperate with each other by leaving and retrieving tuples in the distributed tuple spaces. The key objective of anthill is to build robust and adaptive networks of peer-to-peer services (e.g., file sharing) by exploiting the capabilities of ants to re-shape their activity patterns accordingly to the changes in the network structure. Although we definitely find the idea interesting and promising, a more general flexible approach would be needed to support – other than adaptive resource sharing – adaptive coordination in distributed applications.

The popular videogame "The Sims" [Sims] exploits sorts of computational fields, called "happiness landscapes" and spread in the virtual city in which characters live, to drive the movements of non-player characters. In particular, non-player characters autonomously move in the virtual Sims city with the goal of increasing their happiness by climbing the gradients of specific computational fields. For instance, if a character is hungry, it perceives and follows a happiness landscape whose peaks correspond to places where food can be found, i.e., a fridge. After having eaten, a new landscape will be followed by the character depending on its needs. Although sharing the same inspiration, "Sims' happiness fields" are static and generated only by the environment. In TOTA, instead, tuples are dynamic and can change over time, and agents themselves are able to inject tuples to promote a stronger self-organization perspective.

The MMASS formal model for multi-agent coordination, described in [BanMS02], represents the environment as a multi-layered graph in which agents can spread abstract fields representing different kinds of stimuli through the nodes of this graph. The agents' behavior is then influenced by the stimuli they perceive in their location. In fact agents can associate reactions to these stimuli, like in an event-based model, with the add-on of the location-dependency that is associated to events and reactions. The main difference between MMASS and TOTA the application domain: MMASS is mainly devoted to simulation of artificial societies and social phenomena, thus its main implementation is based on cellular automata, TOTA is mainly interested in distributed (pervasive) computing and, accordingly, its implementation is based on real devices forming wireless networks.

The L2imbo model, proposed in [Dav98], is based on the notion of distributed tuple spaces augmented with processes (Bridging Agents) in charge of moving tuples form one space to another. Bridging agent can also change the content of the tuple being moved for example to provide format conversion between tuple spaces. The main differences between L2imbo and TOTA are that in L2imbo, tuples are conceived as "separate" entities and their propagation is mainly performed to let them being accessible from multiple tuple spaces. In TOTA, tuples form distributed data structure and their "meaning" is in the whole data structure rather than in a single tuple. Because of this conceptual difference, tuples' propagation is defined for every single tuple in TOTA, while is defined for the whole tuple space in L2imbo.

6. CONCLUSIONS AND FUTURE WORKS

Tuples On The Air (TOTA) promotes programming distributed applications by relying on distributed data structures, spread over a network as sorts of electromagnetic fields, and to be used by application agents both to extract contextual information and to coordinate with each other in an effective way. As we have tried to show in this paper, TOTA tuples support coordination and self-organization, by providing a mechanism to both enable agents interactions and to represent contextual information in a very effective way.

Despite the fact there are a lot of examples we had been able to realize with TOTA, we still do not have a general engineering methodology or primitive tuples' types on which to build and generalize other kind of applications. However this is not our specific limit, but it is a current general limitation: a general methodology for dealing with bottom up approaches (like the one promoted by TOTA) is still unknown. However, we think that such methodology could be found in the future and for sure, our final goal would be to develop a complete engineering procedure for this kind of model. In pursuing this goal, deployment of applications will definitely help identifying current shortcomings and directions of improvement. In particular our future work will be based on applying the TOTA model, in the development of new applications for sensor networks with a particular interest in those algorithms exploiting ideas taken from manifold geometry [ZamM02]. From a more pragmatic perspective, several issues are still to be solved for our first prototype implementation to definitely fulfill its promises. First, we must compulsory integrate proper access control model to rule accesses to distributed tuples and their updates. Second, much more performance evaluations are needed to test the limits of usability and the scalability of TOTA by quantifying the TOTA delays in updating the tuples' distributed structures in response to dynamic changes.

7. REFERENCES

[Abe00] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, "Amorphous Computing", Communications of the ACM, 43(5), May 2000.

[BabM02] O. Babaoglu, H. Meling, A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems", in Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS '02), Vienna, Austria, July 2002.

[BanMS02] S. Bandini, S. Manzoni, C. Simone, "Space Abstractions for Situated Multiagent Systems", 1st International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna (I), ACM Press, pp. 1183-1190, July 2002.

[Bar97] Y. Bar-Yam. Dynamics of Complex systems. Addison-Wesley, 1997.

[Bar02] L. Barabasi, "Linked", Perseus Press, 2002.

[BelPR01] F. Bellifemine, A. Poggi, G. Rimassa, "JADE - A FIPA2000 Compliant Agent Development Environment", 5th International Conference on Autonomous Agents (Agents 2001), pp. 216-217, Montreal, Canada, May 2001.

[BonDT99] E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence", Oxford University Press, 1999.

[Bro98] J. Broch, D. Maltz, D. Johnson, Y. Hu, J. Jetcheva, "A Perfomance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", ACM/IEEE Conference on Mobile Computing and Networking, Dallas (TX), Oct. 1998.

[But02] W. Butera, "Programming a Paintable Computer", PhD Thesis, MIT Media Lab, Feb. 2002.

[CabLZ02] G. Cabri, L. Leonardi, F. Zambonelli, "Engineering Mobile Agent Applications via Context-Dependent Coordination", IEEE Transactions on Software Engineering, 28(11):1040:1058, Nov. 2002.

[Dav98] N. Davies, et al, "L2imbo: A distributed systems platform for mobile computing", ACM Mobile Networks and Applications, 3(2):143-156, Aug., 1998.

[GelC92] D. Gelernter, N.Carriero "Coordination Languages and Their Significance", Communication of the ACM, 35(2):96-107, Feb. 1992.

[KepC03] J. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1):41-50, Jan. 2003.

[Loo01] D. Loomis, "The TINI Specification and Developer's Guide", http://www.ibutton.com/TINI/book.html

[MamLZ02] M. Mamei, L. Leonardi, M. Mahan, F. Zambonelli, "Coordinating Mobility in a Ubiquitous Computing Scenario with Co-Fields", Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices, AAMAS 2002, Bologna, Italy, July 2002.

[Nag03] R. Nagpal, A. Kondacs, C. Chang, "Programming Methodology for Biologically-Inspired Self-Assembling Systems", in the AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality, March 2003

[ParBS02] V. Parunak, S. Bruekner, J. Sauter, "ERIM's Approach to Fine-Grained Agents", NASA/JPL Workshop on Radical Agent Concepts, Greenbelt (MD), Jan. 2002.

[PicMR01] G. P. Picco, A. L. Murphy, G. C. Roman, "LIME: a Middleware for Logical and Physical Mobility", In Proceedings of the 21st International Conference on Distributed Computing Systems, IEEE CS Press, July 2001.

[Polybot] http://www2.parc.com/spl/projects/modrobots/simulations/index.html

[Poo00] R. Poor, "Gradient Routing in Ad Hoc Networks", http://www.media.mit.edu/pia/Research/ESP/texts/poorieeepaper.pdf.

[Rat01] S. Ratsanamy,, P. Francis, M. Handley, R. Karp, "A Scalable Content-Addressable Network", ACM SIGCOMM Conference 2001, San Diego (CA), ACM Press, Aug. 2001.

[RomJH02] G.C. Roman, C. Julien, Q. Huang, "Network Abstractions for Context-Aware Mobile Computing", 24th International Conference on Software Engineering, Orlando (FL), ACM Press, May 2002.

[RowD01] A. Rowstron, P. Druschel, “Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems”, 18th IFIP/ACM Conference on Distributed Systems Platforms, Heidelberg (D), Nov. 2001.

[SerD02] D. Servat, A. Drogoul, “Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence?”, AAMAS, Bologna (I), July, 2002.

[SerR02] G. Di Marzo Serugendo, A. Romanovsky, “Designing Fault-Tolerant Mobile Systems”, Proceedings of the International Workshop on scientific engineering of Distributed Java applications (FIDJI'02), Keynote Paper, volume 2604, LNCS, pp. 185-201, Springer-Verlag, 2002

[SheS02] W. Shen, B. Salemi, P. Will, “Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots”, IEEE Transactions on Robotics and Automation 18(5):1-12, Oct. 2002.

[Sims] http://thesims.ea.com

[YimZD02]M. Yim, Y. Zhang, D. Duff, “Modular Robots”, IEEE Spectrum Magazine, February 2002.

[ZamM02] F. Zambonelli, M. Mamei, “The Cloak of Invisibility: Challenges and Applications”, IEEE Pervasive Computing, 1(4):62-70, Oct.-Dec. 2002.

[ZamP02] F. Zambonelli, V. Parunak, “From Design to Intention: Signs of a Revolution”, 1st Intl. ACM Conference on Autonomous Agents and Multiagent Systems”, Bologna (I), July 2002.